The Report Committee for Melody Liao
Certifies that this is the approved version of the following report:

A numerical study of single-machine multiple-recipe predictive maintenance

APPROVED BY
SUPERVISING COMMITTEE

_____
John J. Hasenbein, Supervisor

_____
Erhan Kutanoglu

A NUMERICAL STUDY OF SINGLE-MACHINE MULTIPLE-RECIPE PREDICTIVE MAINTENANCE

by

MELODY LIAO, B.A.

REPORT

Presented to the Faculty of the Graduate School of
The University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science in Engineering

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2011

ACKNOWLEDGMENTS

I would like to sincerely thank my supervisor, Professor John Hasenbein, for giving me the opportunity to work on this topic and for guiding me with patience and wisdom. I am also grateful to Professor Erhan Kutanoglu for serving as a reader. Lastly, I would like to thank the SMART Scholarship and the Graduate School at the University of Texas at Austin for their support.

A NUMERICAL STUDY OF SINGLE-MACHINE MULTIPLE-RECIPE PREDICTIVE MAINTENANCE

by

MELODY LIAO, M.S.E.

THE UNIVERSITY OF TEXAS AT AUSTIN, 2011
SUPERVISOR: JOHN J. HASENBEIN

Effective machine maintenance policy is a critical element of a smooth running manufacturing system. This paper evaluates a multiple-recipe predictive maintenance problem modeled using a M/G/1 queueing system. A numerical study is performed on an optimal predictive maintenance policy. A simulated job-based maintenance policy is used as a baseline for the optimal policy. We investigate the effects of varying degradation rates, holding costs, preventive maintenance times, and preventive maintenance costs. We also examine a two-recipe problem.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Machine maintenance is an integral aspect of an efficient manufacturing system. Preventive maintenance policies have been studied for decades. These policies consider the timing of two types of maintenances: preventive maintenance and corrective maintenances. Corrective maintenance is more costly and time intensive than preventive maintenance, but only occurs when a machine fails. A good preventive maintenance policy considers the trade-offs between more frequent preventive maintenances and the more expensive corrective maintenances. Traditionally, maintenance policies have only considered when maintenance was last performed on a machine. Thus, many of the conventional policies take one of two forms: calendar-based or job-based. In calendar based systems, a preventive maintenance (PM) is performed after a certain amount of time elapsed since the last maintenance. If the machine fails prior to the next scheduled PM, a corrective maintenance (CM) is performed. In job-based systems, a PM is performed after a certain amount of jobs has been processed since the last maintenance. Again, if a machine fails prior to the next PM, then a CM is performed.

However, calendar- and job-based policies may not be ideal. A maintenance policy that only considers the number of jobs processed or the amount of time elapsed can lead to unnecessary maintenances. For example, if a machine remains in good condition despite having gone beyond the required number of jobs or recommended time, performing a PM would be inefficient. Conversely, it is costly when a CM must be done on a machine that fails soon after a maintenance. A policy which considers a more accurate metric of machine status can lead to a policy more custom tailored to a machine's maintenance requirements. New technology has allowed for the more accurate tracking of a machine's condition.

Maximizing machine availability and minimizing production cycle time have become a priority so as to minimize stock on hand and operate "lean." Thus, work-in-progress (WIP) level should be considered in maintenance decisions to ensure a smooth production line and minimize the occurrence of bottlenecks. The ignorance of WIP level is detrimental: When the machine is in a good state and the WIP level is high, a PM would be unnecessary.

Or, on the other hand, an opportunity to do a PM may be missed when machine state is poor and the WIP level is low. Due to both the inefficiencies of calendar- and job-based policies and the lower costs in monitoring machine states, predictive maintenance policies are increasingly becoming a topic of interest.

This report investigates and compares an M/G/1 single-machine job-based policy to the predictive maintenance policy developed by Cai [2]. By creating a simulation which applies a job-based policy to Cai's model, we can directly compare the difference in average total cost between the two policies. Section 2 provides a literature review for related models. Section 3 provides a model description using semi-Markov decision processes (SMDP). Section 4 provides a description of the simulation and numerical investigations.

2. LITERATURE REVIEW

Much research has been done on maintenance scheduling. However, we confine this literature review to the research closely tied to the model used in this report. We examine maintenance models which use a two-dimensional state space, featuring two factors: machine status and WIP level.

Kaufman and Lewis [3] examine M/G/1 single-machine systems where process time may increase as the machine deteriorates. The machine deteriorates one state at a time in random time intervals. Two models are evaluated: repair and replacement. The repair model has random repair times with a positive mean. The replacement model has an instantaneous repair time. It is shown that the optimal policy is monotone in the machine state. However, the optimal policy is not necessarily monotone in the WIP level. Kaufman and Lewis' model [3] is similar to Cai's model [2] in that the two models both use the same state space. However, Kaufman and Lewis use a different degradation mechanism and only investigate the one product, or recipe, problem.

Yao's dissertation [4] integrates a two-dimensional state space to develop a preventive maintenance policy, considering both a "technical state" and the WIP level (Chapter 3 in [4]). The "technical state" is defined to be the number of jobs processed since the last maintenance. Yao [4] develops a control limit policy under certain conditions, where a control limit policy is one which sets a threshold that distinguishes for which states a PM is necessary given the WIP level.

Yao [3] and Cai's [1] models are centered on two dimensional state space models but diverge in their choice of variables. Both share WIP level as a variable, but Yao's model [4] uses the number of jobs processed since last maintenance and Cai's model [2] uses a machine status indicator. In Yao's model [4], the machine has a conditional probability of failing based on the number of jobs processed since the last maintenance, where the machine's time to failure is a discrete random variable. Cai's model [2] allows for random degradation after each processed job, allowing machine status to not be affected at all, to degrade by single or multiple states, or to fail completely.

The work in this report adopts the model developed by Cai [2]. An M/G/1 predictive maintenance policy for both single-product and multiple-product scenarios is developed. An optimal control-limit policy is found. Structural results are proven and numerical examples are provided.

3. METHODOLOGY

3.1 AVERAGE COST SINGLE-RECIPE SMDP MODEL

We adopt Cai's model [2] and simulate a job-based policy to find an optimal policy. We consider a machine modeled by an $M/G/1$ queueing system with an unreliable server, which produces a single product, or recipe. A single recipe problem is equivalent to a queueing model with one class of jobs. The machine produces one product. The machine condition is represented by a machine state, *s.* The machine condition is scaled into $M+1$ possible states. That is, the machine state s is an element of the set $S = \{0,1,2,\dots,M\}$. Machine state 0 is equivalent to a new machine condition and machine state M is equivalent to a machine failure. For simplicity, it is assumed that the machine degrades to a lower state only after a job has been processed.

In this $M/G/1$ system, jobs enter the system according to a Poisson process with rate λ. Jobs are then processed one at a time. The service times are independent and identically distributed random variables following a general distribution. After each job is processed, the machine degrades from machine state r to machine state s $(s \geq r)$ with conditional probability, $q_{r,s}$.

After a machine has completed a job, the system controller must decide its next action. If the machine fails, or reaches machine state M, then the machine must undergo a CM. If no queue exists, then the machine must choose between two actions: wait and do nothing or perform a PM. If a queue exists and the machine state is not M, the machine must choose to either process a job or perform a PM. While the machine is undergoing maintenance, no jobs may be processed. Similarly, while the machine is processing a job, the machine cannot undergo maintenance. After a CM or PM, the machine condition returns to new and the machine state, s, is returned to 0.

The objective of this system is to minimize the average total cost. The system is charged holding costs for each job in the system per time unit. Furthermore, each maintenance

performed, PM or CM, also has a cost associated with it. This problem can be modeled as a semi-Markov decision process.

The decision epochs occur after the processing of each job or after a PM or a CM. If the queue is empty, then an additional decision epoch occurs once the next job arrives, before it is processed. Cai's policy [2] evaluates the WIP level, w, and the machine state, s. The system state is then defined to be (w, s) and the system state space is $\mathbb{Z}^+ \times S$, where $\mathbb{Z}^+$ denotes the set of nonnegative integers. The following notation is used in the model.

w: nonnegative WIP level (including job in process)
s: machine state
S: set of machine states
u_t: action chosen at decision epoch t
$u_{w,s}$: action chosen in state (w, s) in a stationary policy
$U_{w,s}$: set of available actions in state (w, s)
q_{rs}: conditional transition probability of the machine state from r to s $(s \geq r)$
$p_{(w,s)(w',s')}$: transition probability from state $(w, s)$to state (w', s')
τ_c: time for a CM, a continuous random variable with cumulative distribution function $G_c(\cdot)$
τ_p: time for a PM, a continuous random variable with cumulative distribution function $G_p(\cdot)$
τ_a: time to process a job, a continuous random variable with cumulative distribution function $G_a(\cdot)$
$T(u)$: expected duration of action $u \epsilon U_{w,s}$
λ: arrival rate of incoming jobs
p_k^t: probability k jobs arrive in t time units
h: holding cost per job per time unit in the system (in queue or in process)
c_p: cost for a PM
c_c: cost for a CM
$I(A)$: indicator variable for an action A

Actions are denoted as follows. If the machine is in state M, the only available action is:

C: perform a CM if the machine is in state M.

If the machine is not in state M, then the available actions are:

P: perform a PM,

A: continue processing the next job if the queue is not empty, or

W: wait for the next job if the queue is empty.

Thus the state-dependent action space is:

$$U_{w,s} = \begin{cases} \{P, A\}, & \text{if } w > 0 \text{ and } s < M \\ \{P, W\}, & \text{if } w = 0 \text{ and } s < M \\ \{C\}, & \text{if } s = M. \end{cases}$$

A total average cost function is considered in the form

$$\lim_{N\to\infty} E\left\{\frac{1}{t_N}\int_0^{t_N} g(w_t, s_t, u_t)dt\right\},$$

where w_t is the WIP at time t, s_t is the machine state at time t, and t_N is the time of the Nth decision epoch. The function g is given by

$$g(w_t, s_t, u_t) = hw_t + c_p \cdot \delta(u_t = P) + c_c \cdot \delta(u_t = C),$$

where $\delta(\cdot)$ is the Dirac delta function, a generalized function depending on a parameter such that it is zero for all values of the parameter except when the parameter satisfies a given condition, at which point the function takes the value of infinity. Furthermore, the integral of the Dirac delta function over the reals is equal to one.

For average cost problems, the so-called dual LP below gives the optimal average cost:

$$\text{minimize} \sum_{w\in\mathbb{Z}^+}\sum_{s\in S}\sum_{u\in U_{(w,s)}} x(w,s,u)\, y(w,s,u)$$

$$\text{subject to} \sum_{u\in U_{(w',s')}} x(w',s',u) = \sum_{w\in\mathbb{Z}^+}\sum_{s\in S}\sum_{u\in U_{(w,s)}} x(w,s,u)p_{(w,s)(w',s')} \text{ for all } w' \in \mathbb{Z}^+, s' \in S$$

$$\sum_{w\in\mathbb{Z}^+}\sum_{s\in S}\sum_{u\in U_{(w,s)}} x(w,s,u) = 1$$

where $x(w,s,u)$ are the decision variables and $y(w,s,u) = hw + \frac{c(u)}{\tau_u} + \frac{h}{2}\sum_{k\in\mathbb{Z}^+} kp_k^{\tau_u}$. Note $p_k^{\tau_u} = \frac{e^{-\lambda\tau_u}(\lambda\tau_u)^k}{k!}$. The distribution of variable $x(w,s,u)$ represents the long-run distribution of time spent in state (w,s) given action u is taken at state (w,s). The variable $y(w,s,u)$ represents the average per time unit cost of being in state (w,s), accounting for the holding cost and maintenance cost. The last term in the equation represents the holding costs for newly arrived entities while action u is taken. The optimal policy can be obtained directly from the optimal values of the decision variable.

A job-based policy will be used as a baseline for comparison to Cai's policy [2]. In a job-based policy, the system will determine its next action based on the number of jobs the machine has processed since its last maintenance. If the determined threshold has been reached, then a PM will occur. Otherwise, the machine will continue processing jobs.

3.2 Average Cost Two-Recipe SMDP Model

The model is extended to consider a machine which can process two product types or recipes. In realistic applications, the different products may have different effects on the system. For example, one recipe may cause the machine to degrade faster or may cost more to process. The two recipes are called recipe A and recipe B. The two recipes are identical in arrival distribution. In the two-recipe case, the policy considers the WIP level of recipe A and recipe B and the machine state to recommend the next action. If no jobs are in queue, then the policy suggests either waiting or performing a PM. If only recipe A jobs are in queue, then the policy recommends either processing a recipe A job or performing a PM. If only recipe B jobs are in queue, then the policy recommends either processing a recipe B job or performing a PM. If both recipe A and recipe B jobs are in queue, then the policy recommends either processing a recipe A job, processing a recipe B job, or performing a PM. And if the machine state has degraded to state *M*, then the machine must undergo a CM. Here, again, Cai's policy [2] evaluates the WIP level of the recipe A and recipe B jobs and the machine state to recommend an action. A job-based policy will only consider the number of jobs processed since the last maintenance. If both recipe A and recipe B jobs are in the queue and the system decides to continue processing, the jobs will be served in a first-in-

first-out manner. A job-based policy is at a disadvantage here. Because a job-based policies neglect to consider the number of recipe A and recipe B jobs in queue, the average total cost yielded is likely to be suboptimal. For example, if one recipe is cheaper and quicker to process, this recipe should almost always be prioritized to process first to minimize the holding cost. The two-recipe SMDP model uses much of the same notation as the one-recipe SMDP model. Much of the notation can be carried over from the previous section. However, additional notation is required for the two-recipe case:

w_a: nonnegative recipe A WIP level (including the job in process, if applicable)
w_b: nonnegative recipe B WIP level (including the job in process, if applicable)
$q(r, s, u)$: the conditional transition probability from state r to state s,
after processing a recipe u job where $u \in \{A, B\}$

We define $w = w_a + w_b$. Actions are denoted as follows. If the machine is in state M, the only available action is:

C: perform a CM.

If the machine is not in state M, then the available actions are:

P: perform a PM,
A: continue processing the next recipe A job if the queue is not empty,
B: continue processing the next recipe B job if the queue is not empty,
W: wait for the next job if the queue is empty.

Thus the state-dependent action space is:

$$U_{w,s} = \begin{cases} \{P, A, B\}, & \text{if } w_a, w_b > 0 \text{ and } s < M \\ \{P, A\}, & \text{if } w_a > 0 \text{ and } s < M \\ \{P, B\}, & \text{if } w_b > 0 \text{ and } s < M \\ \{P, W\}, & \text{if } w = 0 \text{ and } s < M \\ \{C\}, & \text{if } s = M. \end{cases}$$

The components of the average total cost of the two-recipe machine system are similar to the single-recipe machine system. The total cost is composed of a holding cost,

maintenance cost, and processing cost. Processing cost, which only appears in the two-recipe case, is charged for each job processed. The optimal policy and optimal cost can be found with a linear program similar to the program in the previous section.

$$\text{minimize} \sum_{w\in\mathbb{Z}^+}\sum_{s\in S}\sum_{u\in U_{(w_a,w_b,s)}} x(w_a,w_b,s,u)\, y(w_a,w_b,s,u)$$

subject to

$$\sum_{u\in U_{(w_a\prime,w_b\prime,s')}} x(w_a',w_b',s',u) = \sum_{w_a\in\mathbb{Z}^+}\sum_{w_b\in\mathbb{Z}^+}\sum_{s\in S}\sum_{u\in U_{(w_a,w_b,s)}} x(w_a,w_b,s,u)p_{(w_a,w_b,s)(w_a',w_b',s')} \text{ for all } w' \in \mathbb{Z}^+, s' \in S$$

$$\sum_{w_a\in\mathbb{Z}^+}\sum_{w_b\in\mathbb{Z}^+}\sum_{s\in S}\sum_{u\in U_{(w_a,w_b,s)}} x(w_a,w_b,s,u) = 1$$

where $x(w_a,w_b,s,u)$ are the decision variables and $y(w_a,w_b,s,u) = h(w_a+w_b) + \frac{c(u)}{\tau_u} + \frac{h}{2}\sum_{k\in\mathbb{Z}^+}\sum_{l\in\mathbb{Z}^+}(k+l)\frac{e^{-2\lambda\tau_u}(\lambda_a\tau_u)^k(\lambda_b\tau_u)^l}{k!l!}$. The distribution of variable $x(w_a,w_b,s,u)$ represents the long-run distribution of the time spent in state (w_a,w_b,s) given action u is taken. The variable $y(w_a,w_b,s,u)$ represents the average cost of being in state (w_a,w_b,s), accounting for the holding cost and maintenance cost. The last term in the equation represents the holding costs for newly arrived entities while action u is taken.

4. EXPERIMENTATION

Cai [2] developed a linear program model for the one and two recipe problems in the context of SMDP's. Although a linear programming model can be developed to solve a countable state space SMDP, it is much simpler to solve a finite state SMDP. In any case, the WIP level in any real maintenance problem is always finite. Thus, the state space is simplified so that only a finite number of entities, N, can exist in the system at any given time. Furthermore, processing times are assumed to be deterministic. PM and CM times are also assumed to be deterministic. Note, however, that Cai's model [2] does not require deterministic processing and maintenance times. We compare Cai's policy to a simulated job-based policy. The linear program is executed in Xpress. Cai's code for the one-recipe and two-recipe linear programs can be found in Appendices A and B.

The job-based policy based on Cai's model is simulated using Rockwell Software's ARENA program. A simulation model was built to reflect the mechanics of the system. Appendix C presents a diagram of the simulation for the single-recipe problem. Appendix D presents a diagram of the simulation for a two-recipe problem. Each case was run for 200,000 time units and over 40 repetitions. For each case, several job-count thresholds were tested. The best threshold is the threshold which yields the lowest average total cost. Due to sampling error, choosing the best policy based on simulation point estimates does not guarantee that such a threshold is optimal. However, the variance of the point estimates were generally small, and therefore the best costs for job-based policies we report should still provide a relatively accurate representation of the optimal job-based policy. Note that in the job-based policy simulation, we do not limit the number of entities allowed in the system, although it is rare that the number of entities exceed N. In the single-recipe problem, we used $N = 30$.

The optimal policy and the best job-based policy will be given for each case. Furthermore, long-run average total cost will also be given for the optimal policy and a point estimate and 95% confidence interval for the long-run average total cost will be given for the simulated job-based policy.

4.1 One-recipe Problem

The arrival probabilities for a finite state space SMDP are different from the arrival probabilities for a countable state space SMDP. Thus the model in this section will use arrival probabilities of a truncated Markov chain. At the time of a decision epoch, let j be the number of jobs which enter the system after the current decision epoch and before the next decision epoch. Let $\hat{p}_{j,w}^{t}$ be the probability that j jobs enter the system within t time units given that the initial WIP level is w. The truncated arrival distribution is given below.

$$\hat{p}_{j,w}^{t} = \begin{cases} \dfrac{e^{-\lambda t}(\lambda t)^{j}}{j!}, & 0 \le j < n \\ \left(1 - \displaystyle\sum_{k=0}^{j-1} \frac{e^{-\lambda t}(\lambda t)^{k}}{k!}\right), & j + w = n \\ 0, & j + w > n. \end{cases}$$

Thus the transition probabilities in a countable state space model are also different from the transition probabilities of a finite state model.

4.1.1 Numerical Study

The following parameter values are used as the base setting:

- $N = 30$; $M = 10$; $\lambda = 0.1$ jobs per time unit
- Processing time=6 time units, PM time=7 time units, CM time=30 time units
- Holding cost=\$0.05 per job per time unit; PM Cost=\$0; CM cost=\$20
- The degradation structure: $\forall r, s \in S$ and $s \ge r$,

$$q_{rs} = \begin{cases} \dfrac{1}{N}, & r = 0, 0 \le s \le N - 1 \\ p_{deg}, & r > 0, r = s \\ \dfrac{1 - p_{deg}}{M - s}, & r > 0, r < s \\ 0, & o.w. \end{cases}$$

  where p_{deg} is the probability that the machine state stays in its current level after processing a job. In the base setting, $p_{deg} = 0.9$.

Note that the models were run with N=30, however, because of round-off errors in Xpress, the optimal policy is only given for system states which have a higher long-run probability. Thus the optimal policy diagrams are truncated and show only the policy for WIP levels of up to 20.

In the base setting, Cai's optimal policy [2] yields an average total cost of $0.11 per time unit. The job-based model yields an average total cost of $0.1742 with a 95% confidence interval (CI) of (0.1718, 0.1766). In this scenario, the optimal cost is about 42% less than the job-based policy cost. The best job-based policy had a threshold of nine jobs. That is, after the machine undergoes a PM or CM, the machine processes nine jobs before the next PM, unless the machine fails, at which point, the machine undergoes a CM.

4.1.1.1 Holding Cost

Figure 1 shows the effect of different holding costs from $0.05 to $0.2 with increments of $0.05. The optimal policy has subtle variations as the holding cost increases. In particular, when the machine state is eight and the WIP level is between 1 and 6, inclusive, PM's are performed more sparsely as the holding costs increases. When the holding cost is 0.05 or 0.1, PM's are performed for all WIP levels up to 20 while the machine state is 8. However, when the holding cost increases to 0.15, the policy recommends to continue processing rather than to perform a PM for system states (2,8) and (3,8). When the holding cost is increased to 0.2, processing jobs take priority when the machine state is 8 and the WIP level is between 1 and 6, inclusive. Furthermore, at low WIP levels, PM's are performed at lower machine states while the machine is in better condition. The policy takes advantage of the low or nonexistent holding costs to recondition the machine.

Table 1 shows the effect of the holding cost on the best job-based policy threshold. For the job-based policy, as the holding cost increase from 0.05 to 0.10 to 0.15, more jobs are processed between maintenances. The job-based threshold remained the same when the holding cost increases from 0.15 to 0.2. In increasing the job-based threshold, the policy is undertaking the additional risk that the machine will fail.

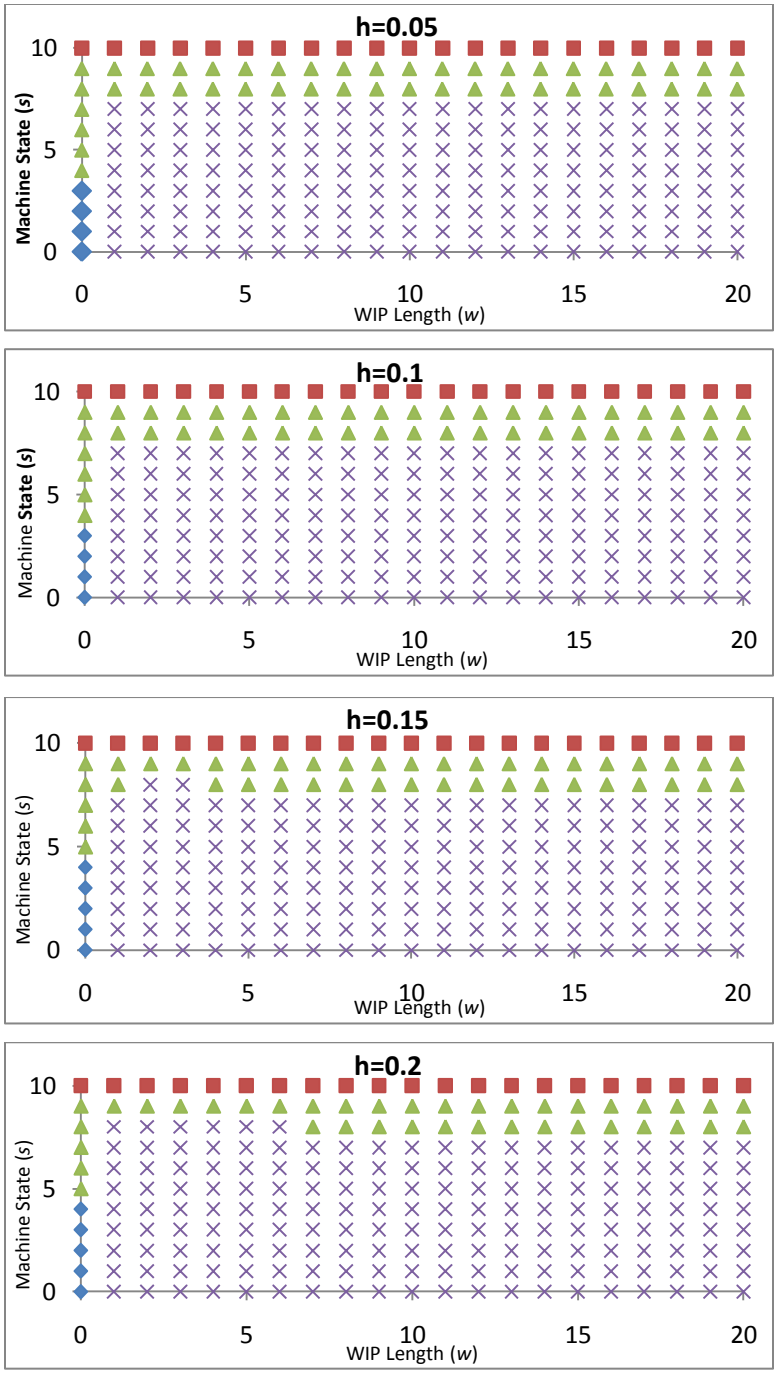


Figure 1. Effect of holding cost on the single-recipe problem
× process; ◆ waiting; ▲ PM; ■ CM

Holding Cost	Job-Based Threshold
0.05	9
0.10	12
0.15	14
0.20	14

Table 1: Effect of holding cost on the job-based policy threshold in the single recipe problem

The average total cost corresponding to the optimal policy and the 95% CI and point estimate for the average total cost corresponding to the job-based policy are listed in Table 2. As the holding cost increases, the difference between the average total costs of the two policies diminishes. The difference in maintenance costs between the two policies decreases relative to the holding cost. This highlights the difference in the long-run average WIP levels of the two policies.

Holding Cost	Job-Based Policy Average Total Cost Point Estimate	Job-Based Policy Average Total Cost (95% CI)	Optimal Policy Average Total Cost	% Difference Point Estimate	% Difference (95% CI)
0.05	0.1742	(0.1718, 0.1766)	0.1103	36.70%	(35.82%, 37.55%)
0.10	0.2821	(0.2784, 0.2859)	0.1933	31.47%	(30.55%, 32.39%)
0.15	0.3884	(0.3833, 0.3936)	0.2762	28.89%	(27.93%, 29.83%)
0.20	0.4946	(0.4879, 0.5014)	0.3576	27.70%	(26.70%, 28.68%)

Table 2: Effect of holding cost on the average total cost in the single-recipe problem

As the holding cost increases, the optimal policy has greater incentive to keeping the WIP level low. Therefore processing jobs take a greater priority over PM's. With a greater holding cost, the optimal system will allow more failures at the expense of the frequency of PM's.

4.1.1.2 Degradation Rate

Figure 2 shows the effect of different degradation structures on the optimal policy. Table 3 shows the effect of the degradation rate, p_{deg}, on the best job-based policy threshold. The degradation rate is varied from 0.7 to 0.9 with increments of 0.1. A large p_{deg}

signifies that the machine is not likely to change from its current machine state or that the machine does not degrade very frequently during the processing of jobs. A small p_{deg} signifies that the machine is likely to change machine states at the completion of a job or that the machine degrades quickly. Thus, a machine with low p_{deg} degrades quickly and requires more frequent maintenances.

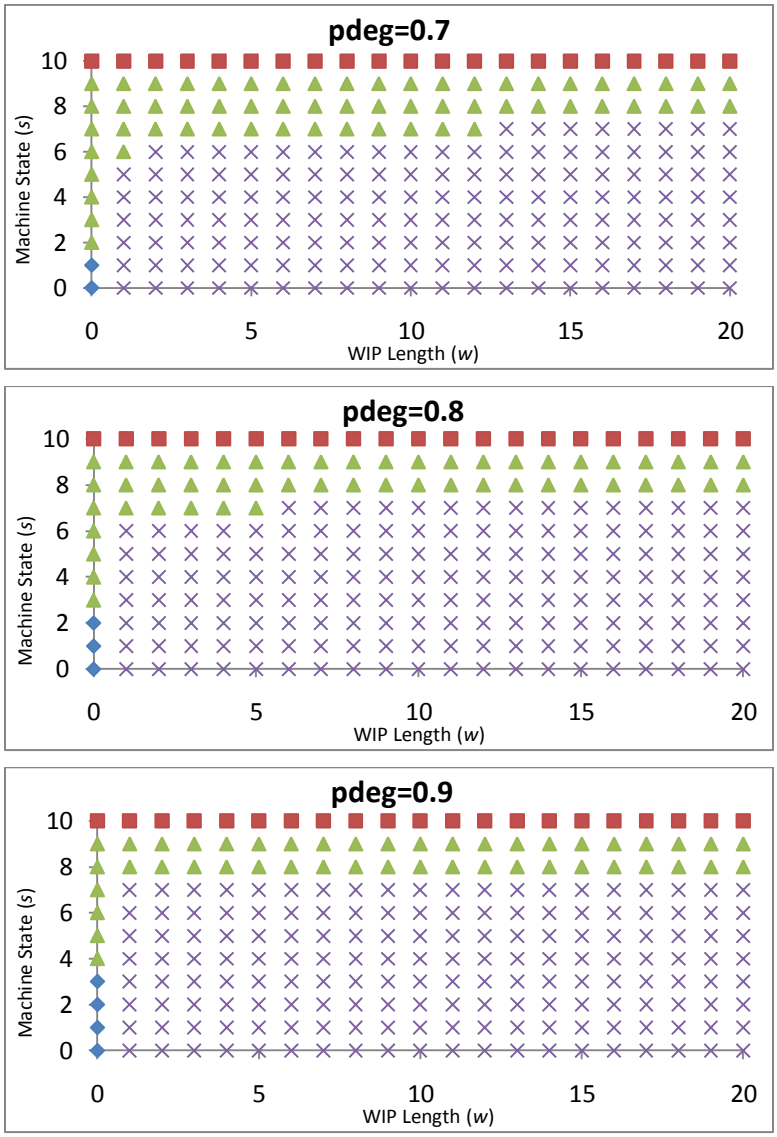


Figure 2. Effect of the degradation rate on the single-recipe problem
× process; ◆ waiting; ▲ PM; ■ CM

p_{deg}	Job-Based threshold
0.7	6
0.8	6
0.9	9

Table 3: Effect of degradation rate on the job-based policy threshold in the single-recipe problem

As p_{deg} decreases, the system is more cautious about how far the machine is allowed to degrade before the next PM. This is also true for the job-based policy when p_{deg} increases from 0.7 to 0.8, here the job-based policy threshold increases from 6 to 9. As p_{deg} decreases, fewer jobs are processed between maintenances and maintenances occur more frequently. This is very evident when machine state is 7. When p_{deg} is 0.7 and machine state is 8, PM's are performed when WIP level is between 0 and 11 inclusive. However, when p_{deg} is 0.8 and machine state is 8, PM's are only performed when WIP level is between 0 and 5 inclusive. Finally, when p_{deg} is 0.9 and the machine state is 8, PM's are only performed when WIP level is 1.The average total cost corresponding to the optimal policy and the 95% CI and point estimate for the average total cost corresponding to the job-based policy are listed in Table 4. The difference between the average total costs of the optimal policy and job-based policy becomes more significant as p_{deg} decreases.

p_{deg}	Job-Based Policy Average Total Cost (95% CI)	Job-Based Policy Average Total Cost Point Estimate	Optimal Policy Average Total Cost	% Difference Point Estimate	% Difference (95% CI)
0.7	0.8129	(0.7761, 0.8493)	0.2727	66.87%	(64.87%, 67.89%)
0.8	0.3447	(0.3370, 0.3523)	0.1789	48.08%	(46.90%, 49.21%)
0.9	0.1742	(0.1718, 0.1766)	0.1103	36.70%	(35.82%, 37.55%)

Table 4: Effect of degradation rate on the average total cost in the single-recipe problem

The less stable the machine condition, the more beneficial it is to know the machine state. As p_{deg} decreases, more uncertainty exists in the system and the job-based policy becomes much less effective than the optimal system.

4.1.1.3 PM Cost

Figure 3 shows the effect of different PM costs on the optimal policy. The PM cost is varied from 0 to 3 in with increments of 1. Table 5 shows the effect of the PM cost on the best job-based policy threshold. The PM Cost is varied from $0 to $3 with increments of $1. In the optimal policy, as PM cost increases, PM's are performed less often when no jobs are in queue. When PM cost is 0, the machine state must be between 3 and 9, inclusive, to perform a PM. When PM cost is 1, the machine state must be between 4 and 9, inclusive, to perform a PM. When PM cost is 2, the machine state must be between 5 and 9, inclusive, to perform a PM. Finally, when PM cost is 3, the machine state must be between 6 and 9, inclusive, to perform a PM. As the PM cost increases, when the WIP level is 0, the machine state must be increasingly higher. As the PM cost increases the threshold for the best job-based policy also increases. Similar effects occur for the optimal policy. For the optimal policy, as the PM cost increases, PM's are performed in higher machine states.

PM Cost	Job-Based threshold
0	9
1	12
2	17
3	19

Table 5: Effect of PM cost on the job-based policy threshold in the single-recipe problem

The average total cost corresponding to the optimal policy and the 95% CI and point estimate for the average total cost corresponding to the job-based policy are listed in Table 6. As the PM cost increases, the difference between the average total costs of the two policies decreases.

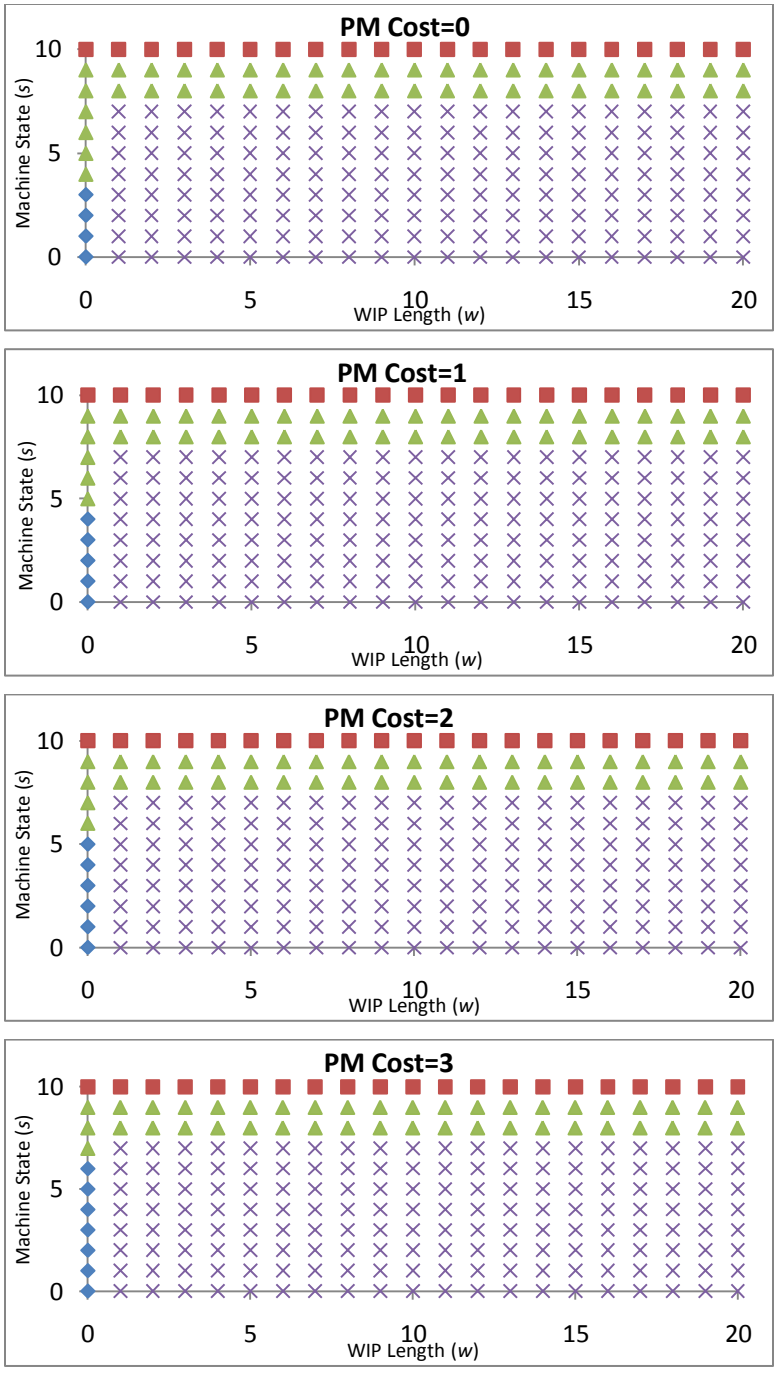


Figure 3. Effect of the PM cost on the single-recipe problem
× process; ◆ waiting; ▲ PM; ■ CM

PM Cost	Job-Based Policy Average Total Cost Point Estimate	Job-Based Policy Average Total Cost (95% CI)	Optimal Policy Average Total Cost	% Difference Point Estimate	% Difference (95% CI)
0	0.1742	(0.1718, 0.1766)	0.1103	36.70%	(35.82%, 37.55%)
1	0.1810	(0.1787, 0.1833)	0.1224	32.37%	(31.50%, 33.24%)
2	0.1862	(0.1838, 0.1885)	0.1316	29.31%	(23.41%, 30.17%)
3	0.1896	(0.1874, 0.1918)	0.1395	26.44%	(25.56%, 27.28%)

Table 6: Effect of PM cost on the average total cost in the single-recipe problem

As the cost of a PM increases, the marginal benefit of performing PM's decrease. Thus, PM's occur less frequently, increasing the likelihood of machine failure.

4.1.1.4 Time for PM

Figure 4 shows the effect of different PM times on the optimal policy. The time for a PM is varied from 5 to 11 in increments of 2. Table 7 shows the effect of the PM time on the best job-based policy threshold. As the PM time increases, the threshold for the best job-based policy also increases. In fact, the job-based threshold sees an increase with every PM time increment. In the case of the optimal policy, the difference among the different policies is most evident when the WIP level is 0. When PM time is 5 and the WIP level is 0, the policy recommends waiting as opposed to performing CM if the machine state is a value between 0 and 2, inclusive. When the PM time is 7 and the WIP length is 0, the policy recommends waiting when the machine state is between 0 and 3, inclusive. When the PM time is 9 or 11 and the WIP length is zero, the policy recommends waiting if the machine state is between 0 and 4, inclusive. Thus, in both policies, PM's occur less frequently in cases with higher PM times.

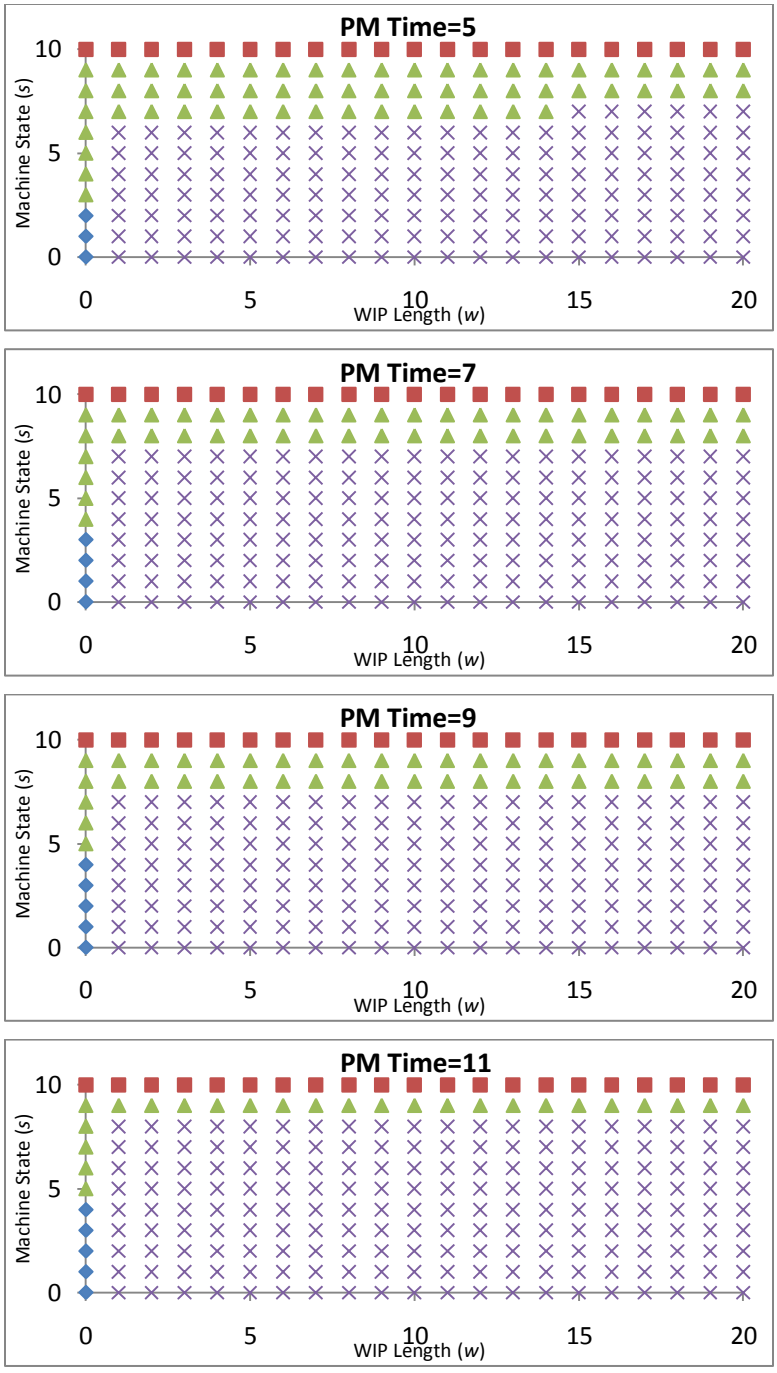


Figure 4. Effect of the PM time on the single-recipe problem
× process; ◆ waiting; ▲ PM; ■ CM

PM time	Job-Based threshold
5	6
7	9
9	12
11	15

Table 7: Effect of PM time on the job-based policy threshold in the single-recipe problem

The average total cost corresponding to the optimal policy and the 95% CI and point estimate for the average total cost corresponding to the job-based policy are listed in Table 8. As the PM time increases, the difference between the average total costs of the job-based policy and the optimal policy decreases. As the difference between the PM time and CM time decreases, the marginal benefit of performing a PM over a CM also decreases.

PM Time	Job-Based Policy Average Total Cost Point Estimate	Job-Based Policy Average Total Cost (95% CI)	Optimal Policy Average Total Cost	% Difference Point Estimate	% Difference (95% CI)
5	0.1643	(0.1619, 0.1665)	0.1017	38.09%	(37.21%, 38.94%)
7	0.1742	(0.1718, 0.1766)	0.1103	36.70%	(35.82%, 37.55%)
9	0.1800	(0.1776, 0.1824)	0.1185	34.17%	(34.17%, 33.27%)
11	0.1844	(0.1820, 0.1868)	0.1257	31.83%	(30.92%, 32.70%)

Table 8: Effect of PM time on the average total cost in the single-recipe problem

The effect of the PM time increasing is similar to the effect of the cost for a PM increasing. However, the effect of increasing PM time is more indirect. As the PM time increases, the cost of a PM increases indirectly through the holding cost. Given the same policy and a longer PM time, jobs will spend more time in queue and yield a higher holding cost.

4.2 TWO-RECIPE PROBLEMS

Again, in a truncated state-space, transition probabilities are different from transition probabilities in a countable state-space. The maximum WIP level is n, so that $w_a + w_b \leq n$. Let i_a and i_b denote the WIP levels of recipe A and recipe B, respectively, at the start of an

action. Let j_a and j_b denote the WIP levels of recipe A and recipe B, respectively, at the completion of an action. Thus the truncated state space for the WIP level is defined as

$$\widetilde{W} = \{(i_a, i_b) | i_a, i_b \in + \text{ and } i_a + i_b \leq n\}.$$

The truncated state space for the two recipe problem is

$$\tilde{S} = \{(i_a, i_{b,}, s) | s \in S, (i_a, i_b) \in \widetilde{W}\}.$$

Let k_a and k_b represent the number of recipe A and recipe B jobs, respectively, that arrive while action u is being performed, where $u \in U_{(i_a, i_b, s)}$.

$$k_a = \begin{cases} j_a - i_a + 1, & u = A, i_a > 0 \\ j_a - i_a, & \text{otherwise} \end{cases}$$

$$k_b = \begin{cases} j_b - i_b + 1, & u = B, i_b > 0 \\ j_b - i_b, & \text{otherwise.} \end{cases}$$

Define $k = k_a + k_b$. The conditional truncated arrival distribution is

$$\hat{p}^u_{(i_a,i_b)(j_a,j_b)} = \begin{cases} \dfrac{e^{-2\lambda\tau_u}(\lambda\tau_u)^k}{k_a!\, k_b!}, & 0 \leq j_a + j_b < n \\ \left(1 - \displaystyle\sum_{l=0}^{k-1} \frac{e^{-2\lambda\tau_u}(2\lambda\tau_u)^l}{l!}\right) C_k^{k_a} \left(\frac{1}{2}\right)^k, & j + w = n \\ 0, & j + w > n. \end{cases}$$

The derivation can be found in [2] and will not be repeated here.

4.2.1 NUMERICAL STUDY

In the two-recipe case, we consider recipe A and recipe B. For simplicity, the two processes have the same arrival rate; however, they have differing degradation rates. The base parameter values are as follows:

- $N = 20$; $M = 10$; $\lambda_A = \lambda_B = 0.05$ jobs per time unit
- Processing time=6 time units, PM time=7 time units, CM time=40 time units
- Holding cost=\$0.05 per job per time unit; PM Cost=\$0; CM cost =\$20

- Process A costs \$0.5 per job; process B costs nothing per job
- The degradation structure: $\forall s, r \in S$ and $s \geq r$,

$$q(r,s,u) = \begin{cases} \frac{1}{N}, & r = 0, 0 \leq s \leq N-1 \\ p_{deg}^{u}, & r > 0, r = s \\ \frac{1 - p_{deg}^{u}}{M - s}, & r > 0, r < s \\ 0, & o.w. \end{cases}$$

where $u \in \{A, B\}$ and p_{deg}^{A}=0.7, p_{deg}^{B}=0.9.

In this base setting, Cai's optimal policy yields an average total cost of \$0.21 per time unit. The best job-based policy yields an average cost of \$0.6944 with a 95% confidence interval of (0.6679, 0.7209). In this scenario, the optimal cost is about 69% less than the job-based policy cost. The best job-based policy has a threshold of six jobs.

4.2.1.1 Holding Costs

Figure 5 and Figure 6 show the effect of different PM times on the optimal policy. The holding cost is varied from 0.05 to 0.2 in increments of 0.05. In the optimal policy, recipe B jobs more often take precedence over recipe A jobs, i.e. when both products are in queue, recipe B jobs are more often processed first. This is logical as recipe A jobs are more costly to process and are more likely to cause the machine to degrade. However, when the total WIP level ($w = w_a + w_b$) is high, recipe A jobs often takes precedence, even when recipe B jobs are available. Furthermore, similar to the one-recipe problem, when the holding cost increases from 0.05 to 0.1, PM's are performed at fewer WIP level states (w_a, w_b) when the machine state is 6. Table 9 shows the effect of the holding cost on the best job-based policy threshold. When the holding cost increases from 0.05 to 0.10, the job-based threshold increases from 6 to 9.

Holding Cost	Job-Based Threshold
0.05	6
0.10	9
0.15	9
0.20	9

Table 9: Effect of holding cost on the job-based policy threshold in the two-recipe problem

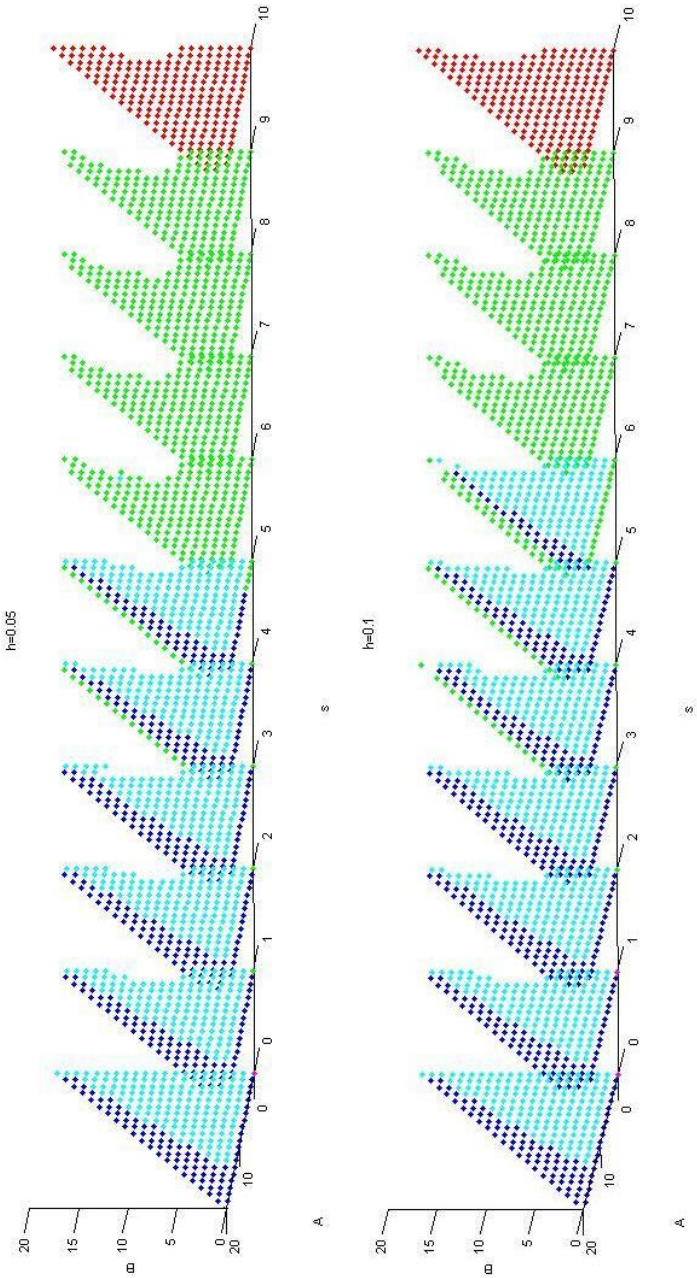


Figure 5. Effect of the holding cost on the two-recipe problem
◆ process A; ◆ process B; ◆ waiting; ◆ PM; ◆ CM

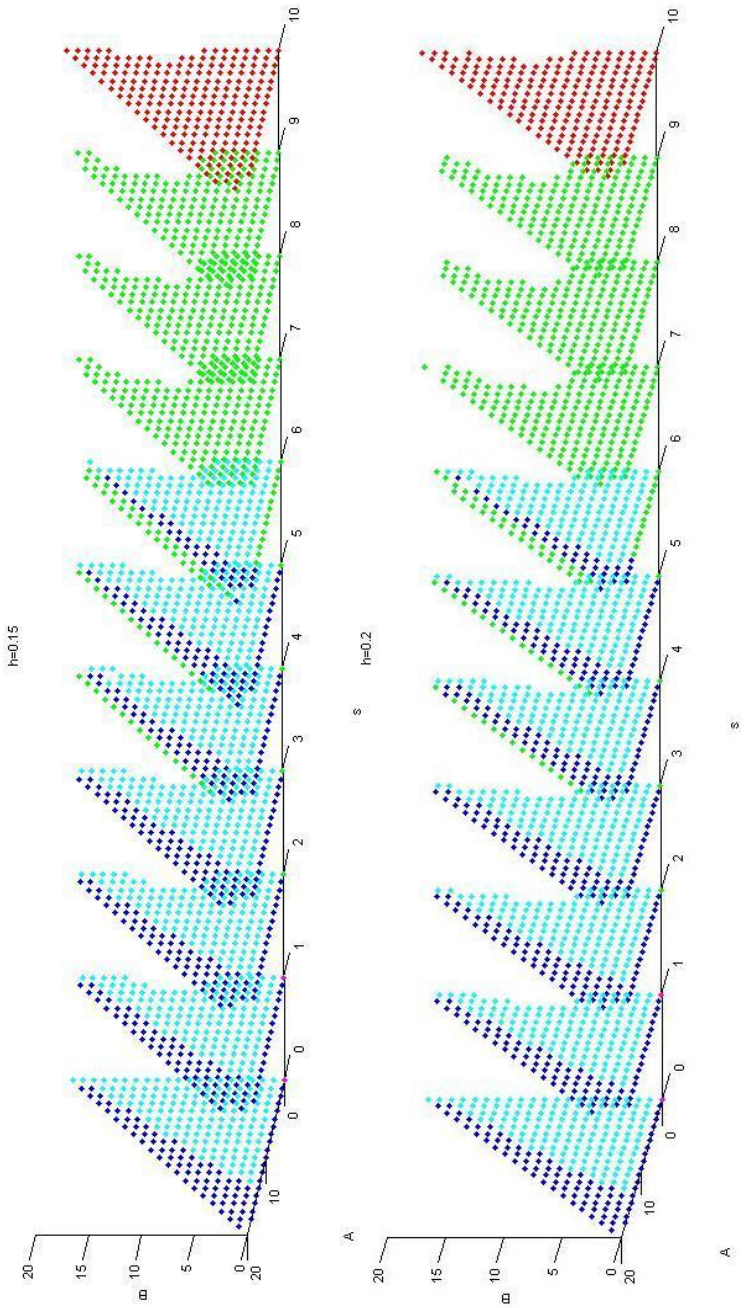


Figure 6. Effect of the holding cost on the two-recipe problem
◆ process A; ◆ process B; ◆ waiting; ◆ PM; ◆ CM

The average total cost corresponding to the optimal policy and the 95% CI and point estimate for the average total cost corresponding to the job-based policy are listed in Table 10. As the holding cost increases, the difference between the average total costs of the job-based policy and the optimal policy also increases.

Holding Cost	Job-Based Policy Average Total Cost Point Estimate	Job-Based Policy Average Total Cost (95% CI)	Optimal Policy Average Total Cost	% Difference Point Estimate	% Difference (95% CI)
0.05	0.6944	(0.6679, 0.7209)	0.2128	69.35%	(68.14%, 70.48%)
0.10	1.1878	(1.1331, 1.2424)	0.3353	71.77%	(70.41%, 73.01%)
0.15	1.6687	(1.5871, 1.7502)	0.4569	72.62%	(71.21%, 73.89%)
0.20	2.1497	(2.012, 2.2380)	0.5785	73.09%	(71.66%, 74.38%)

Table 10: Effect of holding cost on the average total cost in the two-recipe problem

The increase in the difference of the average total costs of the two policies highlights the difference in long-run average WIP levels. If the two policies yielded systems with comparable long-run average WIP levels, then the difference in costs would decrease.

5. CONCLUSION

The numerical study performed in this report confirms the value of Cai's predictive maintenance policy [2]. His optimal policy leads to significant cost savings over a job-based policy. The most significant cost savings occur in multiple-recipe machine systems or systems with a high degradation rate. Furthermore, by exploring varying values for preventive maintenance cost, preventive maintenance time, degradation rate, and holding cost, we see the effect of these values on the dynamics of the system and policies. Increasing PM time, PM cost, and holding cost lead to policies with PM's which occur less frequently. Decreasing the degradation rate leads to policies with PM's that occur more frequently. This report investigated the effects of these parameters, one parameter at a time. However, changing several parameters may yield mixed results. For instance, raising both CM cost and PM cost would yield a smaller change than just raising PM cost, since the change in the trade-offs between performing CM's and performing PM's are more slight than just changing the CM cost or PM cost. On the other hand, if CM cost increased and PM cost decreased, then the optimal policy would likely require PM's to be performed for more system states. Here, performing a PM becomes less costly, especially when compared to the increased CM cost.

APPENDICES

APPENDIX A. CAI'S XPRESS CODE FOR THE ONE-RECIPE CASE

```
model AvgC_Single_Recipe      !average cost single-recipe problem
!In this version, PM is NOT allowed when machine is new (s=0)
uses "mmxprs"; !gain access to the Xpress-Optimizer solver

!optional parameters section
parameters
      maxN = 30               !maximum number of jobs allowed in the
system
      maxM = 10               !number of degradation level
      lambda = 0.1            !arrival rate
      Tc = 6                  !processing time
      Tp = 7                  !PM time
      Tr = 30                 !CM (repair) time
      h = 0.05                !holding cost
      Cp = 0                  !PM cost
      Cr = 20                 !CM cost
      prof = 0                !profit of producing one product
      pdeg = 0.9              !probability of stay in the current
machine status
end-parameters

!sample declarations section
declarations
      Ns = 0..maxN
      Ms = 0..maxM
      As = {"PM","CM","C","W"}            !"C"=Continue
      X: array (Ns,Ms,As) of mpvar  !decision variables in linear
programming

      fac : array(Ns) of real !used to store factorial values
      T: array(As) of real          !expected time for each action
      C: array(As) of real          !cost (-profit) for each action
      delta: array(As) of real      !delta=1 if a=C; delta=0 o/w
      p_hat: array (As, Ns, Ns) of real   !truncated arrival
distribution p_hat(a,i,j) is the probability that arrival of (j-i) jobs
when action a is chosen
      !p_hat(a,i,maxN) = 1 - sum(j in i..(maxN-1))p_hat(a,i,j)
      q:    array (As, Ms, Ms) of real          !transition probability
from machine status s to machine status j for action u
      policy: array (Ns,Ms) of string     !used to record optimal
unichain policy
      auxg: array (Ns,Ms,As) of real      !single stage cost of the
auxiliary problem
end-declarations

writeln("maxN= ",maxN)
writeln("maxM= ",maxM)

T("PM") := Tp
T("CM") := Tr
T("C")  := Tc
T("W")  := 1/lambda
C("PM") :=Cp
C("CM") :=Cr
C("C")  :=-prof
C("W")  := 0
forall(a in As) do
```

```
      if(a="C") then
            delta(a):=1
      else
            delta(a):=0
      end-if
end-do

!calculating factorial values
forall (i in Ns) fac(i) := 1
forall (i in Ns | i>1)  fac(i) := fac(i-1)*i

!calculating truncated arrival distribution in the auxiliary problem
forall(a in As) do
      forall(i in Ns, j in Ns |j<maxN) do
            if (a="C" and j>=(i-1)) then
                  p_hat(a,i,j):= exp(-lambda*T(a))*(lambda*T(a))^(j-
i+1)/fac(j-i+1)
            elif (a="W") then
                  if(i=0 and j=1) then
                        p_hat(a,i,j):=1
                  else
                        p_hat(a,i,j) := 0
                  end-if
            elif(j>=i) then
                  p_hat(a,i,j) := exp(-lambda*T(a))*(lambda*T(a))^(j-
i)/fac(j-i)
            else
                  p_hat(a,i,j) := 0
            end-if
      end-do
      forall(i in Ns|a<>"W") p_hat(a,i,maxN) := 1 - sum(j in 0..(maxN-
1)) p_hat(a,i,j)
end-do

forall(a in As, i in Ns, s in Ms) do
      if (a="W") then
            auxg(i,s,a):=0
      elif(i>0 and a="C") then
            auxg(i,s,a) := h*i + C(a)/T(a) + sum(j in Ns | j>=(i-
delta(a)))h*(j-i+delta(a))*p_hat(a,i,j)/2
      else
            auxg(i,s,a) := h*i + C(a)/T(a) + sum(j in Ns | j>=(i-
delta(a)))h*(j-i+delta(a))*p_hat(a,i,j)/2
      end-if
end-do

!define degradation distribution
forall(u in As, s in Ms, r in Ms) do
      if(u="C" and s<maxM) then
            if (s > r) then q(u,s,r):=0
            !(!revised so that q(0,maxM)=0, i.e., when machine is new
(s=0), no way to fail
            elif (s=0) then
                  if (r=maxM) then q(u,s,r) := 0
                  else q(u,s,r) := 1/(maxM-s)
                  end-if
            !!)
            elif (s=r) then q(u,s,r) := pdeg
```

```
            else q(u,s,r) := (1-pdeg)/(maxM - s)
            end-if
      else
            if (u="W" and s=r and s<maxM) then q(u,s,r) := 1
            elif (u="PM" and r=0 and s<maxM) then q(u,s,r) := 1
            elif (u="CM" and r=0 and s=maxM) then q(u,s,r) := 1
            else q(u,s,r) := 0
            end-if
      end-if
end-do
q("C",maxM, maxM) := 0

!calculating gamma
gamma:=MAX_REAL
forall(a in As) do
      if (a="C") then
            forall(s in Ms|s<maxM) do
                  newGamma:=T(a)/(1-exp(-
lambda*T(a))*lambda*T(a)*q(a,s,s))
                  if (gamma>newGamma) then
                        gamma := newGamma
                  end-if
            end-do
      elif (a="PM") then
                  newGamma := T(a)/(1-exp(-lambda*T(a)))    !for the
case where s=0
                  if (gamma>newGamma) then
                        gamma := newGamma
                  end-if
                  newGamma := T(a)        !for the case where s>0
                  if (gamma>newGamma) then
                        gamma := newGamma
                  end-if
      else
            newGamma:=T(a)
      end-if
      if (gamma>newGamma) then
            gamma := newGamma
      end-if
end-do
gamma := 3/4*gamma

declarations
      p_tilde: array(Ns,Ms,Ns,Ms,As)of real     !transition probability
from (i,s) to (j,r) in auxiliary problem
      action: array(Ns, Ms, As) of real         !action(i,s,a)=1 if "a"
is available for (i,s); 0 o/w
end-declarations

declarations
      test: real
end-declarations

forall(i in Ns, s in Ms, a in As) action(i,s,a) := 1
forall(i in Ns, s in Ms) do
      if (s<maxM) then
            action(i,s,"CM") := 0
            !(!prohibit PM when s=0
```

```
            if(s=0 and i=maxN) then
                  action(i,s,"PM"):=0
            end-if
            !!)
            if(i>0) then action(i,s,"W") := 0
            else action (i,s,"C") := 0
            end-if
      else
            forall(a in As|a<>"CM") action(i,s,a) := 0
      end-if
end-do

forall(i in Ns, s in Ms, j in Ns, r in Ms, u in As | action(i,s,u)=1)
do
      if(s=r and i=j) then
            p_tilde(i,s,j,r,u) :=1 - gamma*(1-
p_hat(u,i,j)*q(u,s,r))/T(u)
      else
            p_tilde(i,s,j,r,u) := gamma*p_hat(u,i,j)*q(u,s,r)/T(u)
      end-if
end-do

forall(i in Ns, s in Ms, a in As|action(i,s,a)=1) do
      test := sum(j in Ns, r in Ms) p_tilde(i,s,j,r,a)
      if ( ((test-1)>1e-7) or ((test-1)<-1e-7)) then
            writeln("EQUAL To ", test, "  at ",i," ",s," ",a)
      end-if
end-do

OBJ:= sum(i in Ns, s in Ms, a in As) action(i,s,a)*X(i,s,a)*auxg(i,s,a)

!constraints
forall(j in Ns, r in Ms) ST(j,r) := sum(a in As)action(j,r,a)*X(j,r,a)
= sum(i in Ns, s in Ms, a in
As)action(i,s,a)*X(i,s,a)*p_tilde(i,s,j,r,a)

ConSum := sum(i in Ns, s in Ms, a in As) action(i,s,a)*X(i,s,a) = 1

exportprob(EP_MIN, "/srv/home/ml28334/Documents/one_Avg_rev", OBJ)

writeln("Begin running model")
minimize(OBJ)
writeln("End running model")

forall(i in Ns, s in Ms, u in As) do
      if (getsol(X(i,s,u))>1e-7) then
            if (policy(i,s)="") then
                  policy(i,s) := u
            else
                  policy(i,s) := policy(i,s) + u
            end-if
      end-if
end-do

writeln("Solution: ", getobjval, "     ")
!output the policy to a file
fopen("/srv/home/ml28334/Documents/single_Ag_rev.dat", F_OUTPUT)
writeln("Solution: ", getobjval, "     ")
```

```
writeln("Ns Ms    As")
forall(i in Ns, s in Ms) writeln(i,"      ",s," ",policy(i,s), "    ")
forall(i in Ns, s in Ms, u in As) writeln(i,"   ",s,"     ",u,"
",getsol(X(i,s,u)))
forall(i in Ns, s in Ms, u in As) writeln(i,"   ",s,"    ",u,"
",getsol(auxg(i,s,u)))
forall(i in Ns, s in Ms, u in As) writeln(i,"   ",s," ",u,"
",action(i,s,u), "    ")
writeln("maxN= ",maxN," maxM= ",maxM,"    lambda= ", lambda)
forall(u in As) write("T(",u,")= ", T(u),"      ")
writeln
forall(u in As) write("Cost(",u,")= ", C(u),"   ")
writeln
writeln("h=", h)
fclose(F_OUTPUT)
end-model
```

APPENDIX B. CAI'S XPRESS CODE FOR THE TWO-RECIPE CASE

```
model ModelName
uses "mmxprs"; !gain access to the Xpress-Optimizer solver
parameters
      maxN = 20
                        !maximum number of jobs allowed in the system
      maxM = 10               !number of degradation level
      lambda = 0.05           !arrival rate
      Tj = 6                  !processing time
      Tp = 9                  !PM time
      Tr = 40                 !CM (repair) time
      h = 0.05                !holding cost
      Cp = 0                  !PM cost
      Cr = 30                 !CM cost
      Ca = 0.5                !cost to process a
      Cb = 0                  !cost to process b
      pdeg_a = 0.7         !probability of staying in the current state
after processing a
      pdeg_b = 0.9          !probability of staying in the current
state after processing b
end-parameters

declarations
      Ns = 0..maxN
      Ms = 0..maxM
      !Rs = {"A","B"}
      As = {"A", "B", "PM", "CM","W"}     !action set

      auxg: array (Ns,Ns,Ms,As) of real   !single stage cost of the
auxiliary problem
      X: array (Ns,Ns,Ms,As) of mpvar     !decision variables in linear
programming

      fac : array(Ns) of real
      T: array(As) of real                !expected time for each
action

      p_hat: array (As,Ns,Ns,Ns,Ns) of real     !truncated arrival
distribution
      q:    array (As, Ms, Ms) of real    !transition probability from
machine status s to machine status j
      TC : array(As, Ns, Ns) of real      !holding cost for incoming
job
      TH : array(As, Ns) of real          !holding cost of current WIP
      Cost: array(As) of real             !cost of action
      policy: array (Ns, Ns, Ms) of string
end-declarations

writeln("maxN= ",maxN)
writeln("maxM= ",maxM)

Cost("A") := Ca
Cost("B") := Cb
Cost("PM") := Cp
Cost("CM") := Cr

T("PM") := Tp
T("CM") := Tr
```

```
T("A")  := Tj
T("B") := Tj
T("W")  := 1/(2*lambda)

!calculating factorial
forall (i in Ns) fac(i) := 1
forall (i in Ns | i>1)  fac(i) := fac(i-1)*i

!calculating truncated arrival probability
forall(u in As) do
      forall(ia in Ns, ib in Ns, ja in Ns, jb in Ns) do
            if(u="A" and ia>0) then
                  ka := ja-ia + 1
                  kb := jb-ib
            elif(u="B" and ib>0) then
                  ka := ja-ia
                  kb := jb-ib + 1
            else
                  ka := ja-ia
                  kb := jb-ib
            end-if
            k := ka+kb
            if(u="W") then
                  if(ia=0 and ib=0 and ( (ja=1 and jb=0) or (ja=0 and
jb=1) ) ) then
                        p_hat(u,ia,ib,ja,jb) := 0.5
                  end-if
            elif((ja+jb)<maxN and ka>=0 and kb>=0) then
                  if( (u="A" and ia>0) or (u="B" and ib>0) or u="PM" or
u="CM") then
                        p_hat(u,ia,ib,ja,jb) := exp(-
2*lambda*T(u))*(lambda*T(u))^k/(fac(ka)*fac(kb))
                  else
                        p_hat(u,ia,ib,ja,jb) := 0
                  end-if
            elif( (ja+jb)=maxN and ka>=0 and kb>=0) then
                  if( (u="A" and ia>0) or (u="B" and ib>0) or u="PM" or
u="CM" ) then
                        p_hat(u,ia,ib,ja,jb) := ( 1-sum(w in 0..(k-
1))exp(-2*lambda*T(u))*(2*lambda*T(u))^w/fac(w)
)*(0.5)^k*fac(k)/fac(ka)/fac(kb)
                  else
                        p_hat(u,ia,ib,ja,jb) := 0
                  end-if
            else
                  p_hat(u,ia,ib,ja,jb) := 0
            end-if
      end-do
      forall(ia in Ns, ib in Ns, ja in Ns, jb in Ns| (ia+ib)>maxN or
(ja+jb)>maxN) p_hat(u,ia,ib,ja,jb) := 0
end-do

forall(u in As, ia in Ns, ib in Ns, s in Ms | (ia+ib)<=maxN) do
      if (u="W") then
            auxg(ia,ib,s,u):=0
      elif(ia>0 and u="A" and s<maxM) then
```

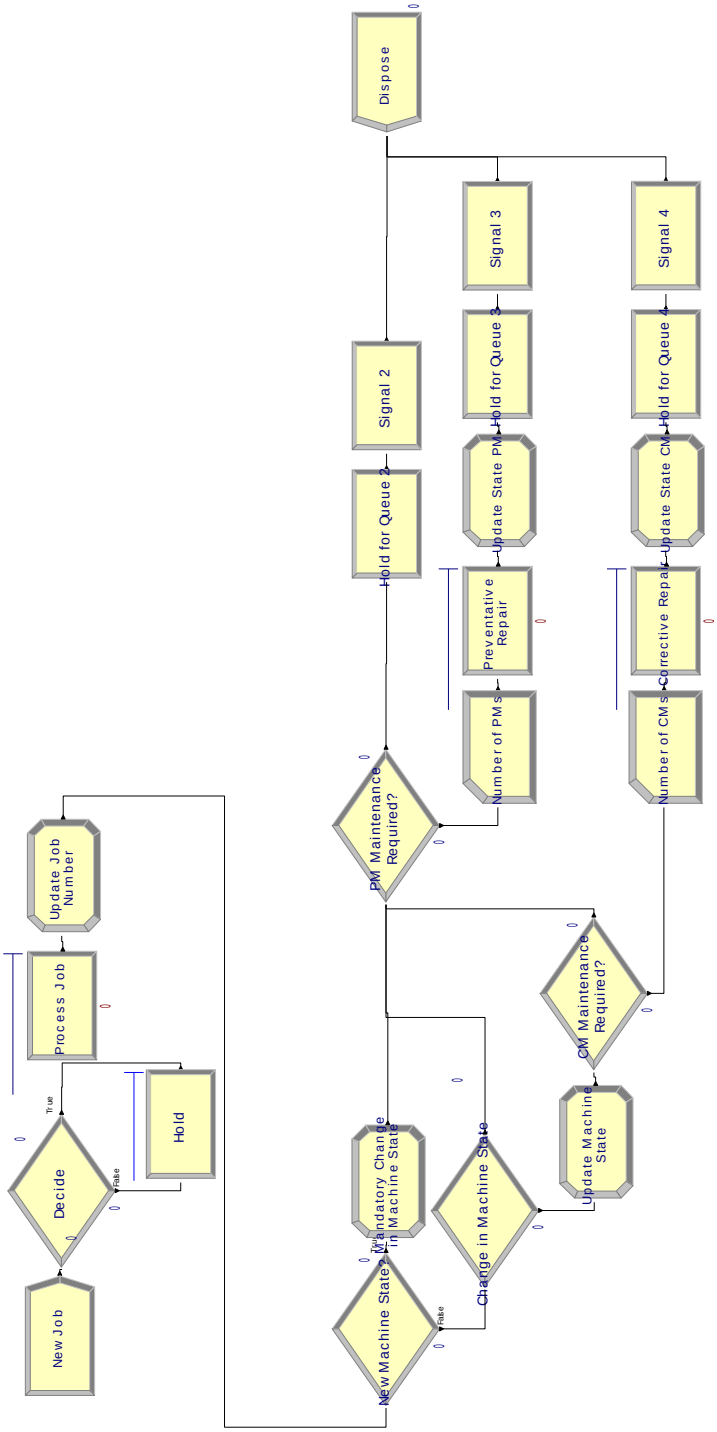
```
            auxg(ia,ib,s,u) := h*(ia+ib) + Cost(u)/T(u) + sum(ja in Ns,
jb in Ns | ja>=(ia-1) and jb>=ib)h*(ja+jb-ia-
ib+1)*p_hat(u,ia,ib,ja,jb)/2
      elif(ib>0 and u="B" and s<maxM) then
            auxg(ia,ib,s,u) := h*(ia+ib) + Cost(u)/T(u) + sum(ja in Ns,
jb in Ns | ja>=ia and jb>=(ib-1) )h*(ja+jb-ia-
ib+1)*p_hat(u,ia,ib,ja,jb)/2
      elif( (u="PM" and s<maxM) or (u="CM" and s=maxM) ) then
            auxg(ia,ib,s,u) := h*(ia+ib) + Cost(u)/T(u) + sum(ja in Ns,
jb in Ns | ja>=ia and jb>=ib) h*(ja+jb-ia-ib)*p_hat(u,ia,ib,ja,jb)/2
      else
            auxg(ia,ib,s,u) := 0
      end-if
end-do


forall(u in As, s in Ms, r in Ms) do
      if( (u="A" or u="B") and s<maxM) then
            if (s > r) then q(u,s,r):=0
            !revised so that q(0,maxM)=0, i.e., when machine is new
(s=0), no way to fail
            !elif (s=0) then
            !     if (r=maxM) then q(u,s,r) := 0
            !     else q(u,s,r) := 1/(maxM-s)
            !     end-if
            elif(u="A") then
                  if(s=r) then
                        q(u,s,r) := pdeg_a
                  else
                        q(u,s,r) := (1 - pdeg_a)/(maxM - s)
                  end-if
            elif(u="B") then
                  if(s=r) then
                        q(u,s,r) := pdeg_b
                  else
                        q(u,s,r) := (1-pdeg_b)/(maxM - s)
                  end-if
            end-if
      else
            if (u="W" and s=r and s<maxM) then q(u,s,r) := 1
            elif (u="PM" and r=0 and s<maxM) then q(u,s,r) := 1
            elif (u="CM" and r=0 and s=maxM) then q(u,s,r) := 1
            else q(u,s,r) := 0
            end-if
      end-if
end-do
q("A",maxM, maxM) := 0
q("B",maxM, maxM) := 0

!calculating gamma
gamma:=MAX_REAL
forall(a in As) do
      if (a="A" or a="B") then
            forall(s in Ms|s<maxM) do
                  newGamma:=T(a)/(1-exp(-
2*lambda*T(a))*2*lambda*T(a)*q(a,s,s))
                  if (gamma>newGamma) then
                        gamma := newGamma
```

```
                  end-if
            end-do
      elif (a="PM") then
                  newGamma := T(a)/(1-exp(-2*lambda*T(a)))  !for the
case where s=0
                  if (gamma>newGamma) then
                        gamma := newGamma
                  end-if
                  newGamma := T(a)        !for the case where s>0
                  if (gamma>newGamma) then
                        gamma := newGamma
                  end-if
      else
            newGamma:=T(a)
      end-if
      if (gamma>newGamma) then
            gamma := newGamma
      end-if
end-do
gamma := 3/4*gamma

declarations
      p_tilde: array(Ns,Ns,Ms, Ns,Ns,Ms,As)of real    !transition
probability from (ia,ib,s) to (ja,jb,r) in auxiliary problem
      action: array(Ns,Ns,Ms,As) of real        !action(ia,ib,s,a)=1 if
"a" is available for (i,s); 0 o/w
end-declarations

forall(ia in Ns, ib in Ns, s in Ms, ja in Ns, jb in Ns, r in Ms, u in
As | (ia+ib)<=maxN and (ja+jb)<=maxN) do
      if(s=r and ia=ja and ib=jb) then
            p_tilde(ia,ib,s,ja,jb,r,u) := 1 - gamma*(1-
p_hat(u,ia,ib,ja,jb)*q(u,s,r))/T(u)
      else
            p_tilde(ia,ib,s,ja,jb,r,u) :=
gamma*p_hat(u,ia,ib,ja,jb)*q(u,s,r)/T(u)
      end-if
end-do

forall(ia in Ns, ib in Ns, s in Ms, u in As) action(ia,ib,s,u) := 1
forall(ia in Ns, ib in Ns, s in Ms, u in As | (ia+ib)>maxN)
action(ia,ib,s,u) := 0
forall(ia in Ns, ib in Ns, s in Ms | (ia+ib)<=maxN) do
      if (s<maxM) then
            action(ia,ib,s,"CM") := 0
            if (s=0) then
                  action(ia,ib,s,"PM") := 0
            end-if
            if(ia>0 or ib>0) then
                  action(ia,ib,s,"W") := 0
                  if(ia=0) then action(ia,ib,s,"A"):=0
                  elif(ib=0) then action(ia,ib,s,"B"):=0
                  end-if
            else
                  action (ia,ib,s,"A") := 0
                  action (ia,ib,s,"B") := 0
            end-if
      else
```

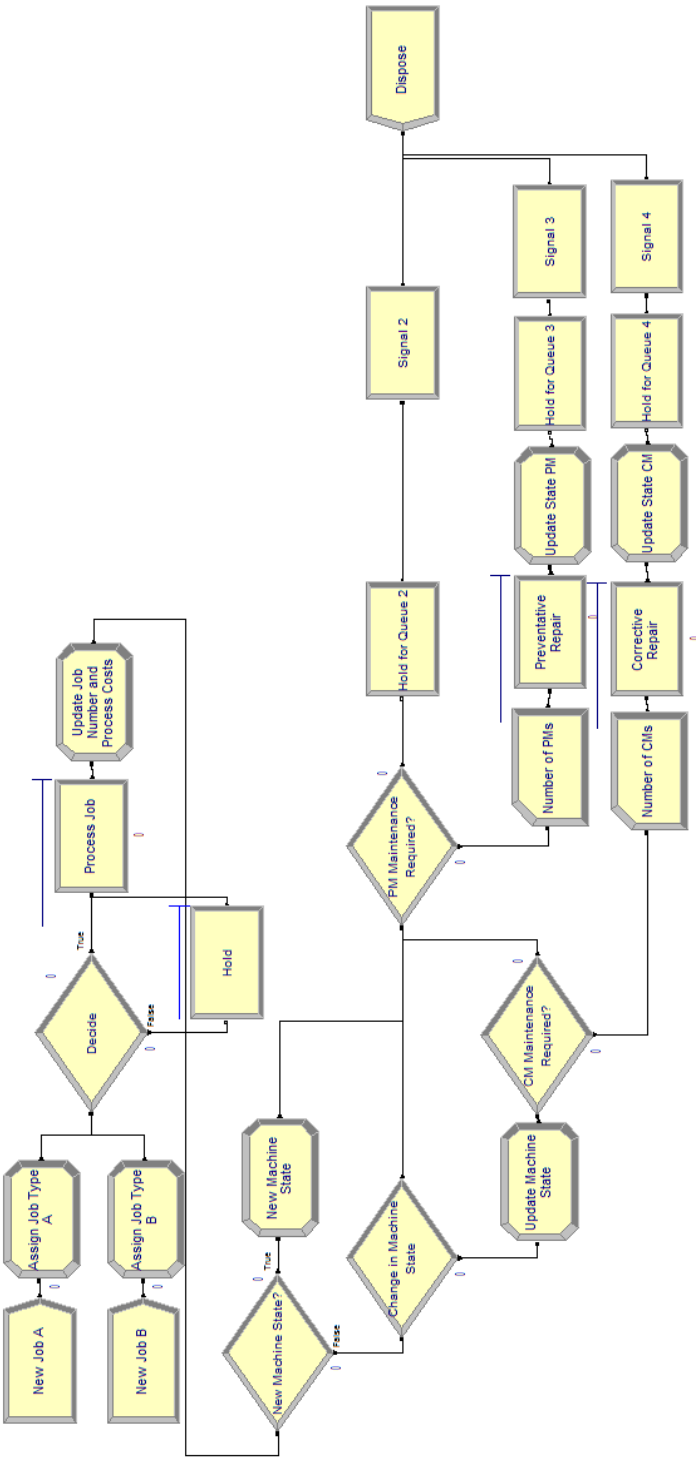
```
            forall(u in As|u<>"CM") action(ia,ib,s,u) := 0
      end-if
end-do

!check the probability sums to 1
flag_1:=true
forall(ia in Ns, ib in Ns, u in As) do
      available := sum(s in Ms) action(ia,ib,s,u)
      if(available=1) then
            test := sum(ja in Ns, jb in Ns) p_hat(u,ia,ib,ja,jb)
            if( (test-1)<-1e-5 or (test-1)>1e-5 ) then
                  writeln ("NOT EQUAL TO ONE AT ", ia," ",ib," ",u,"
Equals to ", test)
                  flag_1 := false
            end-if
      end-if
end-do
if (flag_1=true) then
      writeln("Out transition probability is correct")
else
      flag_1:=true
end-if

OBJ:= sum(ia in Ns, ib in Ns, s in Ms, u in As | (ia+ib)<=maxN)
action(ia,ib,s,u)*X(ia,ib,s,u)*auxg(ia,ib,s,u)

!constraints
forall(ja in Ns, jb in Ns, r in Ms | (ja+jb)<=maxN) ST(ja,jb,r) :=
sum(u in As)action(ja,jb,r,u)*X(ja,jb,r,u) = sum(ia in Ns, ib in Ns, s
in Ms, u in As |
(ia+ib)<=maxN)action(ia,ib,s,u)*X(ia,ib,s,u)*p_tilde(ia,ib,s,ja,jb,r,u)

ConSum := sum(ia in Ns, ib in Ns, s in Ms, u in As | (ia+ib)<=maxN)
action(ia,ib,s,u)*X(ia,ib,s,u) = 1

exportprob(EP_MIN, "/srv/home/ml28334/Documents/two_Avg", OBJ)

writeln("Begin running model")
minimize(OBJ)
writeln("End running model")

forall(ia in Ns, ib in Ns, s in Ms, u in As) do
      if (getsol(X(ia,ib,s,u))>1e-7) then
            if (policy(ia,ib,s)="") then
                  policy(ia,ib,s) := u
            else
                  policy(ia,ib,s) := policy(ia,ib,s) + u
            end-if
      end-if
end-do

writeln("Solution: ", getobjval, "     ")

!output the policy to a file
fopen("/srv/home/ml28334/Documents/two_Ag_rev.dat", F_OUTPUT)
writeln("Solution: ", getobjval, "     ")
writeln("Ns Ns    Ms    As")
```

```
forall(i in Ns, j in Ns, s in Ms) writeln(i,"   ",j," ",s,"
      ",policy(i,j,s))
writeln
writeln("maxN= ",maxN," maxM= ",maxM,"    lambda= ", lambda)
forall(u in As) write("T(",u,")= ", T(u),"      ")
writeln
forall(u in As) write("Cost(",u,")= ", Cost(u),"      ")
writeln("h= ",h)
fclose(F_OUTPUT)


end-model
```

APPENDIX C. SINGLE-RECIPE JOB-BASED PREVENTIVE MAINTENANCE POLICY ARENA SIMULATION



APPENDIX D. TWO-RECIPE JOB-BASED PREVENTIVE MAINTENANCE POLICY ARENA SIMULATION



BIBLIOGRAPHY

[1] D. P. Bertsekas, *Dynamic Programming and Optimal Control Vol 1 & 2*. Belmont, Massachusetts: Athena Scientific, 1995.

[2] Y. Cai. *Semiconductor Manufacturing Inspired Integrated Scheduling Problems: Production Planning, Advanced Process Control, and Predictive Maintenance*. PhD thesis, University of Texas at Austin, 2008.

[3] D. Kaufman and M. E. Lewis. *Machine Maintenance with Workload Considerations*. Naval Research Logistics, Vol. 54(7), 750-766, 2007.

[4] X. Yao. *Optimal Preventative Maintenance Policies for Unreliable Queueing and Production Systems.* PhD thesis, University of Maryland, 2003.

This document does not include the vita page from the original.